Writing HTML

Table of Contents

Text Styling Conventions In These Tutorials

Code

```
Code that can be typed into our documents are styled
like this.
```

Basic Information

The text required for reading and understanding as we go along is styled like this.

Advanced Information

Advanced tips and comments are styled like this. They provide extra information that may be useful, but are not essential to read or understand the section in which they appear.

Browser Display

The web page we would expect to see in our web browser as a result of the code we write is styled like this.

What is HTML?

Pages on the Internet are written in HTML - **HyperText Markup Language**.

Hypertext means that the text can contain links to other pages on the Internet. A reader does not have to simply read a web page from start to finish, in the way that we would normally read a book or magazine. The reader can click on a link and end up somewhere completely different on the Internet. So, the flow of reading is ***non-linear***, that is, it is not a simple straight flow from start to finish.

Markup Language means a language that contains information about itself. By *marking up* text, we give some extra information about the words on the page.

An example in the Real World is the use of a yellow highlighter pen on an essay or article. By highlighting important words and sentences, we effectively *mark up* the text and say, "this piece of text highlighted in yellow is important". Our essay then contains two types of text - text that is important (i.e. that highlighted in yellow) and text that is not so important (i.e. text that is not highlighted).

We may go further - by highlighting our essay in different colours, or underlining words. In this way, we have given some extra information to the page by distinguishing between different types of text.

In HTML, we *mark up* text by the use of ***tags***. These tags give some information about the text they are applied to. They allow us to give the web page more meaning than if it was just text on its own.

Web Standards

What Are Web Standards?

The Web is a growing technology, as is HTML itself. In the early days of the Internet, all sorts of ways were used to write HTML. This was good for experimenting with new ways of achieving what web designers wanted to create, but also meant that different web browsers struggled to display the web pages in the same way as each other.

Nowadays, there are clear ***Web Standards*** for writing HTML. These are important to stick to, so that every browser will display your web pages just how you wanted them to be displayed, and every web designer is using exactly the same language for writing their pages.

HTML Tags

HTML is a collection of tags that have specific meanings. Generally, the tags enclose the text that they are describing, in this way:

```
<tag>Some piece of text.</tag>
```

What is happening in this previous example?

First, we have the ***opening tag***: `<tag>.` The tag starts with a < *(less than)* symbol and ends with a > *(greater than)* symbol. Then we have the piece of text we are describing: `Some piece of text.` Finally, we have the ***closing tag***: `</tag>`. The closing tag is exactly the same as the opening tag, except that it contains a / *(forward-slash)* symbol.

The whole example above – the opening tag, the text contents and the closing tag is collectively called an ***element***.

This is essentially the way we mark up text in HTML and other, related languages. Once we learn a few basic tags, we can start creating HTML documents that can be recognised by web browsers all around the world. In this way, we can publish pages to the World Wide Web.

The Main Structure Of Our HTML Document

<html> HTML Tag

This is the first tag on the page. It tells the ***browser*** (the program that people use to view web pages, e.g. Firefox or Internet Explorer) that this document is written in HTML. There are other languages used on the Internet, so it is essential to tell the browser what language we are using.

We will put `<html>` right at the start of the page, and `</html>` right at the end. Everything else in the document is within this one HTML *element*.

> Actually, as we will learn later, there *is* another tag or two that should come before the HTML tag. That is called the "Doctype" tag and tells the browser what kind of HTML we use in our document. As the HMTL language is constantly evolving, we need to tell the browser what type of HTML we have used and how it should read the document.

<head> Head Tag

This encloses the `head` of the document, which contains extra information about the document itself.

For example, the `head` may contain information about the title of the page, whether the page is written in English or some other language, and it may contain a description and keywords about our page that can be useful for search engines to get an overview of our document.

> We will learn about tags that can come in the **`head`** later (see **`meta`**, **`link`**, **`style`** and **`script`** tags). For now, let's look at the **`title`** tag...

<title> Title Tag

This allows us to give a title to the page, which is usually displayed in the top left-hand corner of the browser. For example:

```
<title>A very great web page</title>
```

<body> Body Tag

The **body** of the page contains all the text, pictures, links and so on – the bit that people actually see when they look at your page.

The Structure Of Our Page So Far

Using the tags we've already looked at, we have the start of a page structure, into which will go all our content – the interesting bits of writing, images, video files, or whatever we want to put into our page.

Here is the structure of our page so far, with some example text:

```
<html>
<head>
    <title>A very great web page</title>
</head>

<body>
    All of our excellent content goes here.
</body>
</html>
```

Simple isn't it?

Try it out by saving this code in your text editor and then opening the file in your browser. You will see a very simple web page. The title of the page will be displayed at the top of the browser window: "A very great web page". The text, "All of our excellent content goes here", will appear in the main body of the page.

So we now need to know the way in which to mark up our content – the bit in between the **body** tags...

Creating A Paragraph

<p> Paragraph Tag

This encloses a paragraph of text on the page.

```
<p>
    We put our paragraph in here.
</p>
```

Strong Tag

This gives the enclosed text added "strength". The default (i.e. the usual) way that this is displayed in our browser is as **bold text**, but can choose to style it however we want to.

Emphasis Tag

This gives added "emphasis" to text. The default style for this is *italic text*.

For example, we may have in between the **body** tags a paragraph like this:

```
<p>
This is the <strong>very first</strong> line on our
new page. I am <em>so very</em> happy.
</p>

<p>
This is another paragraph.
</p>
```

This will produce something like this:

This is the **very first** line on our new page. I am *so very* happy.

This is another paragraph.

Heading Tags

<h1>, <h2>, <h3>, <h4>, <h5>, <h6> Headings 1 to 6

In order to give meaning to the sections within our document, we can give each section a heading. A document would usually have an overall heading at the very top, and lesser headings throughout the document.

The first heading tag in a document will **always** be <h1>. And this tag should only appear once in our document. For example:

```
<h1>All about me</h1>

<p>
This is the <strong>very first</strong> line on our
new page. I am <em>so very</em> happy.
</p>
```

Unless we tell it to otherwise, the browser will style the text within the <h1> tags in large bold letters:

All About Me

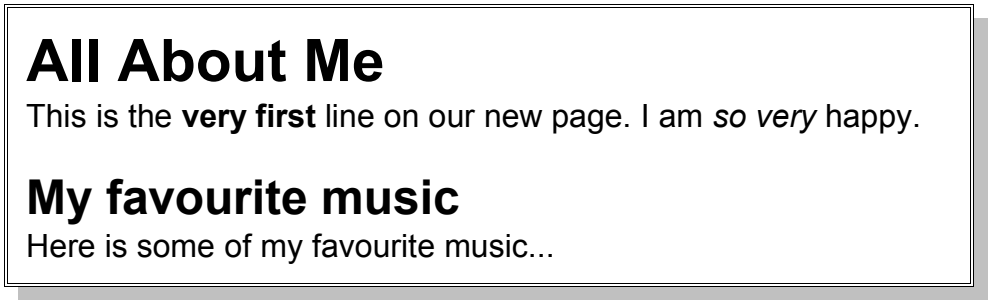
This is the **very first** line on our new page. I am *so very* happy.

When we want to make a sub-header in the document, we use the <h2> tag:

```
<h1>All about me</h1>

<p>
This is the <strong>very first</strong> line on our
new page. I am <em>so very</em> happy.
</p>

<h2>My favourite music</h2>
<p>Here is some of my favourite music...</p>
```



We can keep putting header tags into our document, all the way down to <h6>. In this way, the relative importance of the different sections within our document takes shape. For example:

```
<h1>All about me</h1>

<p>
This is the <strong>very first</strong> line on our
new page. I am <em>so very</em> happy.
</p>

<h2>My favourite music</h2>
<p>Here is some of my favourite music...</p>

<h3>Blues Music</h3>
<p>Blues music is very blue.</p>

<h3>Funk Music</h3>
<p>Funk music is pretty funky.</p>

<h2>My favourite books</h2>
<p>Books are good for reading.</p>

<h3>Fiction</h3>
<p>Everybody likes a good story.</p>

<h3>Non-Fiction</h3>
<p>Not everything is make-believe though.</p>
```

The code would look something like this:

My favourite music

Here is some of my favourite music...

Blues Music

Blues music is very blue.

Funk Music

Funk music is pretty funky.

My favourite books

Books are good for reading.

Fiction

Everybody likes a good story.

Non-Fiction

Not everything is make-believe though.

Self-Closing Tags

 Breaking Space

This tag puts a line break in the flow of text on the page.

Because it does not need to describe or enclose any text, we do not need to write separate opening and closing tags, which might look like this: `<br></br>`.

Instead, the tag can be written so that it is ***self-closing***. That is, it opens and closes all within one tag. It looks like this:

```
<br />
```

The `<br` part of the tag opens the element. The `/>` then closes it. There is a single space between the `<br` and the `/>`.

<hr /> Horizontal Rule

This puts a horizontal line on the page. Like the `<br />` tag, this tag is also *self-closing* because it does not need to describe or enclose any text.

Tag Attributes: Inserting Hyperlinks and Images

<a> Anchor Tag

This is the tag that makes the Internet what it is. It allows us to put ***hyperlinks*** (or simply, "***links***") to other web pages within our text. The links may be to pages within our own website, or to any other website on the Internet.

With this tag, we enclose the text that we want to turn into a hyperlink – i.e. the text that typically a user will click on to activate the link and go to the new page:

```
<p>
If you want to, you can <a>download my music</a> for
free.
</p>
```

The text `download my music` would appear by default as a blue underlined link in the browser:

If you want to, you can download my music for free.

But it is not enough to simply enclose the text that we want the user to click on. We also need to say which web page we want to link to. We need to give the tag some extra information, and we do that by giving the tag an ***attribute***. The general way of writing attributes within tags is like this:

```
<tag attribute="value">Content</tag>
```

In this case, we use the `href` (***hypertext reference***) attribute, giving the web address we want to link to in quotes:

```
<p>
If you want to, you can <a href="music.htm">download
my music</a> for free.
</p>
```

<img> Image Tag

This allows us to put images on our page. We need to say which image we want to use. That is, we need to give the source of the image. We do this with a `src` attribute:

```
<img src="mydog.jpg" />
```

The `src` attribute gives the location of the image in relation to the web document. It may be an image on your computer's hard drive, or it may be an image on a website on the Internet.

In the previous example, we tell the browser to look in the same folder as the web document and then look for the file called `mydog.jpg`

If the image had the same filename, but was contained within a sub-folder called `images`, we would write the code like this:

```
<img src="images/mydog.jpg" />
```

If the image was on another website, we would need to provide the full web address:

```
<img src="http://www.example.com/images/mydog.jpg" />
```

This `img` tag is self-closing because it does not need to enclose any text.

Additionally, we can (and should) provide some descriptive text for those who cannot see images in their browser. This will be useful for those who are blind or have poor eyesight, those using a text-only browser or web-enabled mobile phone, or even a search engine like Google that scans websites to find out what they contain). We do this by using the `alt` attribute, which stands for "alternative text".

```
<img src="mydog.jpg" alt="This is a photo of my
dog" />
```

Unordered Lists and Ordered Lists

What are lists?

Often, we will want to include some kind of list on our page. A list is simply of collection of items of information that are connected by a common theme. Here are some examples of lists:

- My favourite bands
- Presidents of the United States
- Wettest countries of the world
- Things to do before I die

The collection of example lists above is in itself a list.

`<ul>` Unordered List

A unordered list is one in which there is no particular order to the list entries that it contains. For example, "different colours".

`<li>` List Item

We enclose each item in our list in `<li></li>` tags:

```
<p>Some different colours:</p>
<ul>
    <li>red</li>
    <li>yellow</li>
    <li>orange</li>
    <li>pink</li>
    <li>blue</li>
</ul>
```

The previous code would display something like this:

Some different colours:

- red

- yellow
- orange
- pink
- blue

`<ol>` Ordered List

An ordered list is one in which there is an order to the list entries that it contains. That is, the entries would make sense if they were consecutively labelled 1, 2, 3, 4, etc. For example:

```
<p>My greatest musical influences:</p>
<ol>
    <li>PR Sarkar</li>
    <li>Bob Dylan</li>
    <li>Ali Farka Toure</li>
    <li>Van Morrison</li>
    <li>Pink Floyd</li>
    <li>Asian Dub Foundation</li>
</ol>
```

This would display by default like this:

My greatest musical influences:

1. PR Sarkar
2. Bob Dylan
3. Ali Farka Toure
4. Van Morrison
5. Pink Floyd
6. Asian Dub Foundation

Using a List for Site Navigation

On our website, we will have a collection of pages that link together. The pages within that collection will have some kind of relative importance between each other.

We will want to make a ***navigation area*** on each of our web pages, so that the user can quickly navigate their way through our site. This would be a series of links from our web page to other pages.

It makes good sense to make that collection of links into a list. For example:

```
<h2>Site Navigation</h2>
<ul>
    <li><a href="index.htm">Home Page</a></li>
    <li><a href="music.htm">Music</a></li>
    <li><a href="photos.htm">Photos</a></li>
    <li><a href="news.htm"></a>News</li>
    <li><a href="contact.htm">Contact Me</a></li>
</ul>
```

This would display something like this:

Later on, we will learn how to style our pages so that they display in the browser just how we want them to.

By structuring our site navigation as an HTML list, we open up lots of possibilities for controlling the way that it displays. (See **CSS stylesheets**).

<!-- --> Adding Comments To Our Code

We may want to write something in the code of our web page that acts as a comment to ourselves – or to other web designers involved in creating our website. We would not want this comment to be shown in the browser, but simply to remain in the code as a note to ourselves, ignored by the browser.

Some examples are: a note about a future change that we wish to make to the page, a comment about what the section of code near the comment is for so that we can remember its importance, to temporarily prevent a section of code being displayed by the browser while we are writing and testing our page.

We add a comment, or by putting `<!--` in front of the text we wish to comment and putting `-->` after it. For example:

```
<!-- I must remember to add my new track to the music
download page -->

<p>
If you want to, you can <a>download my music</a> for
free.
</p>
```

When we look at CSS stylesheets, we'll see a different method for adding comments to the CSS code. In CSS, we can put a `/*` before the comment text and `*/` after the text.

No Double Dashes Within Comments

It is invalid code to use double dashes, i.e. two successive hyphens ("--"), within a comment. The following examples are **invalid**:

```
<!-- This comment -- is -- invalid -->

<!----------------------------------->
```

However, you could separate the two dashes with a space to make it valid:

```
<!-- This comment - - is - - valid -->

<!-- ===== - - - ===== - - - ===== -->
```

Special Characters & Character Entities

A Problem: The < Character

As we have seen, by placing tags in an HTML document, we tell the browser to do something special with the content in the document. The browser knows that a piece of text is a tag because it starts with a < character.

So, what happens if we want to use the < character as part of the content of our page? For example, we may want the browser to show this:

<< Go back

We **cannot** simply write this code:

```
<p>
    << Go back
</p>
```

The browser would get confused by the line `<< Go back` and would expect the two < symbols to be opening a two new tags. This is because < is a *special character* in HTML. That is, it means something special to the browser and is not treated like other text characters.

We need to use a specific code, called a *character entity*, to represent the < character. That code is **`<`** where the `lt` stands for "less than", the name of the character.

So, the way we would write the code for the example above is this:

```
<p>
    &lt;&lt; Go back
</p>
```

Another Problem: The & Character

Here we stumble across another problem. We have just used the & character not to display an "&" but to tell the browser some information. This means that & is also a *special character*.

As such, whenever we want to display an "&" in the content of our page, we need to use the *character entity* for the & character. This character entity is **`&`** which stands for "ampersand", the name of the character.

For example, if we want to display this in the browser:

> The character entity for the < character is <

We would write the following code:

```
<p>
    The character entity for the &lt; character is
&amp;lt;
</p>
```

The > Character

While it is not essential to encode the > *(greater than)* character, it is recommended that you do. Because the < *(less than)* character must be encoded, it will make your code easier to read if you also encode the > character. The character entity to use is **`>`** (which stands for "greater than") as in the following example:

```
<p>
    An opening body tag looks is written &lt;body&gt;
</p>
```

Which will display in the browser:

> An opening body tag is written <body>

Special Characters And Their Character Entities

<	less than	<
>	greater than	>
&	ampersand	&
"	quot	"

Quoting Text Using The `blockquote` Tag

`<blockquote>` Blockquote Tag

You may wish to use a quote from another source on your web page. For example, you might quote from a book, newspaper, an article from another website or from an email you've received.

It is useful to have a standard tag in which to put quotes. When we get to styling our page, we can then make those quotes display in a particular way. The default display for a blockquote element is for the browser to *indent* the contents on both the left and right sides.

For example:

```
<h1>Some Quotes</h1>

<h2>The Hitchhikers Guide To The Galaxy – Douglas
Adams</h2>

<blockquote>

<p>
<strong>Arthur:</strong> You know, it's at times like
this, when I'm trapped in a Vogon airlock with a man
from Betelguese, and about to die of asphyxiation in
deep space that I really wish I'd listened to what my
mother told me when I was young.
</p>

<p>
<strong>Ford:</strong> Why, what did she tell you?
</p>

<p>
<strong>Arthur:</strong> I don't know, I didn't
listen.
</p>

</blockquote>
```

This would display like this:

Some Quotes

The Hitchhikers Guide To The Galaxy – Douglas Adams

Arthur: You know, it's at times like this, when I'm trapped in a Vogon airlock with a man from Betelguese, and about to die of asphyxiation in deep space that I really wish I'd listened to what my mother told me when I was young.

Ford: Why, what did she tell you?

Arthur: I don't know, I didn't listen.

Citing Your Sources

If the source of the quote is a web address, you can use the `cite` attribute of the `blockquote` tag to specify this source:

```
<blockquote cite="http://www.hitchhikersguide.com">
</blockquote>
```

`blockquote` is a block-level element. That means that all of its contents are contained within a block on the page that is separated from the preceding or following blocks, such as a `p` paragraph or `h1` heading.

To put a quote inline (i.e. inside another block), such as a `p` paragraph element, you can use the **`<q>`** tag. You can also cite the source of the quote inline using the **`<cite>`** tag:

```
<p>
     Then <cite>Arthur</cite> said, <q>I don't know, I didn't listen</q>.
</p>
```

Creating Data Tables in HTML

We may want to display tabular data on our page. For example, budget of the costs involved in setting up a music recording studio:

Item	Cost Per Item	Quantity Required
Computer system	1000	2
Mixing desk	2000	1
Comfy chairs	35	6

We create a table using the **<table>** tag. We then proceed by creating a table row with the **<tr>** tag. In the example above, we have 4 table rows. Its structure in HTML looks like this:

```
<table>

<tr></tr>
<tr></tr>
<tr></tr>
<tr></tr>

</table>
```

In our above example table, we have 3 table cells within each table row. We also have two different types of table cells used in our table. One type is a table header cell, e.g. "Quantity Required", and the other type is a table data cell, e.g. "6".

To create a table header cell, we use the **<th>** tag and to create a standard table data cell, we use **<td>**.

For example:

```
<table>

<tr>
    <th>Item</th>
    <th>Cost Per Item</th>
    <th>Quantity Required</th>
</tr>
```

```
<tr>
     <td>Computer system</td>
     <td>1000</td>
     <td>2</td>
</tr>

<tr>
     <td>Mixing Desk</td>
     <td>2000</td>
     <td>1</td>
</tr>

<tr>
     <td>Comfy chairs</td>
     <td>35</td>
     <td>6</td>
</tr>

</table>
```

Tables Are For Data, Not Layout

HTML tables are intended for displaying tabular data, such as the example above. However, in the days before CSS (*Cascading Style Sheets* - the language we will use to style our pages), web designers used HTML tables to take control of the layout of their displayed pages.

For example, a huge table was created to fill the entire page body, with the site navigation section in one big table cell on the left and the site banner in a cell at the top, etc.

In those times, this was the only way to achieve the layout effects that designers wanted. However, CSS allows many more styling possibilities and is preferrable to using tables for layout for some very good reasons:

- When the page content is split between different cells of an HTML table, it becomes harder for search engines to understand the flow of the document.

- It also becomes harder for a screen reader (a browser that reads web pages, for users with poor eyesight) to follow the document flow.

- It is harder to achieve a good layout in small-screen browsers, such as those on palm-top computers and mobile phones.

- It is more confusing for the web designer to read their own code.

- Tables usually require more code to be written than if we use a CSS stylesheet. This adds a little to the file sizes of our pages, meaning slightly longer downloading times for the pages and higher costs for the hosting of our website.

- The tables approach fails to separate the style of the page from the content of the page. By forcing the page content into an HTML table, we limit the number of ways in which it can be displayed.

- The HTML loses its power as being the *markup language* for the information in our document. Instead, it gets muddied by trying to also style that information.

- In a nutshell, we lose the flexibility that separating style from content allows us. HTML is for *marking up* information, CSS is for *styling* that marked up information.

  This will become clearer once we start learning CSS styling.

Styling HTML Documents With CSS

So far we have been learning how to structure the content of our pages with HTML. We have learned that it is important to separate the information in our HTML from the presentation, or styling, of that content. If we do that, we maintain the maximum flexibility available to us – we can access the content information in many different ways and we can also specify many different ways of styling the content.

We may wish to style our content in various ways depending on where it is to be displayed. For example, we can create a style for viewing the pages on a standard desktop monitor, another style for viewing on a mobile phone, another style for printing the page on a printer, and yet another style for a screen reader to speak the page for a blind user.

For the moment, let's just look at how to create a style for a standard screen. We will write the style in a way that will allow our pages to display well on other screens too – such as palm-tops and mobile phones.

What is CSS?

CSS stands for *Cascading Style Sheets*.

The HTML we have written is like a heirarchy of different elements: The `body` element contains `h1` elements and `p` elements; the `p` elements may contain `a` elements, `img` and `ul` elements; the `ul` elements contain `li` elements that may contain `a`, `strong` or `em` elements, etc.

When we create a style for any element, the style *cascades* down to all the other elements contained within it. This is a powerful technique that makes it very easy to create and modify styles in our pages.

Where To Write CSS Styles

There are three places in which we can specify our styles:

- In the `head` element of each HTML document – the styles specified here will affect all HTML in that document.
- Inside a specific element within the document – the styling will affect only that element.
- In a separate file, called an *external stylesheet*, which we can link to any page on our website – the stylesheet will then affect every page that links to it.

We will usually want to create an external stylesheet. This gives us the maximum flexibility when creating styles for our website. With a single change to a style in the stylesheet, we can change the look of our entire website.

It is not generally advisable or necessary to specify a style within specific elements in the HTML. By doing so, we miss out on the advantages of separating presentation (or styling) from content.

Creating A Style Element

Let us start by learning how to style a single HTML document. We will place the styles in the `head` of the document. If we want, we can later cut and paste these styles into an external stylesheet so that we can affect all the pages on our website.

Inside the `head` element, we place a `style` element. We also need to add an attribute, `type`, with the value `text/css` to tell the browser what kind of style code we are writing:

```
<head>
<title>My Page Title</title>

<style type="text/css">
</style>

</head>
```

All of our CSS code then gets written within the opening and closing tags of the `style` element.

`color`, Our First Style

Here is a typical piece of CSS code. It makes the text within any `h1` elements on the page blue in colour.

```
h1 { color:blue; }
```

Yes... the word "color" is spelled without a "u". All spellings in HTML and CSS is in American English.

Let us look at each part of the CSS code above:

1. We first specify what we want to apply our style to - in this case, the `h1` element.
2. We then open the style information for that element with a `{` *curly brace*.
3. We then write the CSS property that we want to apply – in this case, `color`.
4. We separate the property from its value with a `:` *colon*.

5. We give the property a value, `blue`.

6. We end the `color` statement with a `;` semicolon.

7. Finally, we close the `h1` style with a `}` *curly brace*.

It does not matter whether or not there are spaces or line breaks between any of the words or punctuation in the code. You should write the code so that it is easy for you, the web designer, to read and understand it.

Not Enough Words For The Colours Of The World

In the above example, we gave the `color` property a value of `blue`, which makes text within our web page blue in colour. Colour values such as `blue` can also be used to specify the colour of backgrounds, borders and more.

There are a number of pre-defined colour values like `blue` available in CSS: there is `white`, `black`, `yellow`, `red`, `green` and more. But what do we do if we wish to specify a very particular shade of a particular colour? There are not enough words to describe all the possible colours that exist, so instead we need a way to specify exactly which colour we want...

To explain this properly, we must take a glance at a couple of fundamentals in Physics and Mathematics.

Red, Green & Blue (RGB)

You may have learned in Science at school that when different colours of light are projected onto a white background, they produce new and different colours. For example:

First Light	***Second Light***	***Resultant Light***
Red	Green	Yellow
Green	Blue	Cyan (similar to turquoise)
Blue	Red	Magenta (similar to violet)

When red, green and blue are all present together, we get white light. When no light is present, we get black, which is the absence of colour.

In fact, any colour of light can be created by mixing differing amounts of red, green and blue light. For example, the colour orange is achieved by mixing mostly red light, with a lot of green light, and no blue light.

The stronger the brightness of red, green and blue light, the closer we get to pure white light. And the weaker the brightness, the closer we get to pure black.

This gives us a system for describing any colour – all we do is specify the relative amounts of red, green and blue light present in that colour. We can use this system for specifying our colour values in CSS.

Hexadecimal

In CSS, we use a two-digit *hexadecimal* code to specify the values of red, green and blue light present in a colour. To understand what hexadecimal is, we'll take a brief detour into Mathematics...

In our standard system of numbers, we have 10 different number units:
0, 1, 2, 3, 4, 5, 6, 7, 8, 9.
This number system is called *Base 10*, or *Decimal*.

In decimal, 9 is the highest number in the sequence. If we add 1 to 9, we get 10.

It is possible to use a 6 further number units, to get the number sequence:
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.
Because there are now 16 number units, this is called *Base 16*, or *Hexadecimal*.

In hexadecimal, we count from 0 to F. The highest number in the sequence is F, which equals 15 in decimal.

If we add 1 to F (which would be 16 in decimal), we get 10. The sequence then goes:
10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 1A, 1B, 1C, 1D, 1E, 1F.

The highest number we can express in two digits is FF, which is 255 in decimal. If we include the number 0, that means we can express 256 possible values with a two-digit hexadecimal number.

In CSS, we can use these two-digit hexadecimal numbers to specify 256 different amounts of red, green or blue light present in a colour.

For example:

Red	***Green***	***Blue***	***Resultant Colour***
00	00	00	Black
FF	FF	FF	White
99	99	99	Grey
CC	CC	CC	Light Grey
FF	00	00	Red
00	FF	00	Green
00	00	FF	Blue
FF	FF	00	Yellow
FF	00	FF	Magenta
00	FF	FF	Cyan

Red	***Green***	***Blue***	***Resultant Colour***
03	60	90	Bluey-Green!

Looking at the table above, we see that the colour we have labelled "Bluey-Green" is a mixture of the following values: Red 03, Green 60, Blue 90. These specific RGB values combine to make our specific Bluey-Green colour.

We can write the 3 specific RGB values of 03, 60 and 90 together to make a 6-digit hexadecimal code for the colour: 036090. This 6-digit code specifies "Bluey-Green".

Writing Hexadecimal Colour Codes in CSS

So, finally we get back to the CSS...

This is how we use the two-digit hexadecimal values to specify a colour's Red, Green and Blue (RGB) values:

```
h1 { color:#036090; }
```

We use the # hash character to mark the beginning of our hexadecimal code. The first two digits (03 in this case) give the value of Red. The second two digits (60 in this case) give the Green value. And the last two digits (90 in this case) give the Blue value.

We can use this system for writing colour values for other CSS properties, such as `background-color` and `border-color`:

```
body { background-color:#FFD306; }

p { border-color:#3498F3; }
```

6-Digit and 3-Digit RGB Values

The RGB value #336699 is 6 digits long. There is a way to write some 6 digit RGB values as 3 digit values:

If both digits in a two-digit pair are identical to each other, e.g. 00, 11, 44 and EE, then we can express that number in just one digit, e.g. 0, 1, 4 and E.

If the Red, Green and Blue values are all two-digit identical pairs, then we can write each two-digit pair as just one digit. This reduces the RGB value to 3 digits: one for Red, one for Green and one for Blue.

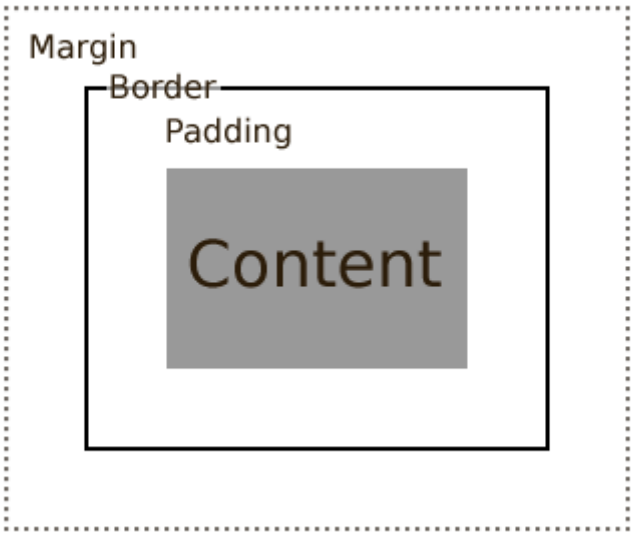
For example, we could write the 6-digit RGB value #336699 as a 3-digit value #369.

In CSS, you can write your colour values in 6 digits or 3 digits - whichever you wish.

The Box Model

`margin`, `padding` & `border` Properties

A *block-level element*, such as a `p` paragraph, can be thought of like a *box*. The box is split up into different parts: the content, the padding, the border and the margin. This structure is known as the *Box Model*:



- **Content** – This is the text, images, and everything else contained within the element.
- **`padding`** – This is the space immediately around the content.
- **`border`** – This is a line on the boundary of the padding.
- **`margin`** – This is the space beyond the border, which touches the boxes of adjacent elements.

Each part of the box can be individually styled using CSS.

The 4 Sides Of The Box

The padding, border and margin of a box can be further split up into their 4 sides:

- top
- right
- bottom
- left

We can specify individual CSS properties for each of the 4 sides.

Padding

You can specifiy an amount of padding for each of the 4 sides using any of the standard CSS *measurement units*, such as `px` *(pixels)*, `em` *(em widths)* and `%` *(percentage of parent element's box)*.

You can specify any of the `padding` CSS properties: `padding-top`, `padding-right`, `padding-bottom`, `padding-left`.

For example:

```
p {
     padding-top:10%;
     padding-right:20%;
     padding-bottom:15%;
     padding-left:25%;
}
```

The above code will make the `p` element's box have padding of differing thicknesses at its top, right, bottom and left sides.

Writing Values For Padding In Shorthand

The previous example code can be simplified by writing a shorthand version that combines each of the 4 sides in a single property, called `padding`:

```
p { padding:10% 20% 15% 25%; }
```

The 4 values of the `padding` property run in the following order: top, right, bottom, left.

If you find this sequence difficult to remember, just think of the sides of the box in terms of the directions of a compass or a map: North, East, South, West *("Never Eat Shredded Wheat")*.

The previous code example and the example before it will have an identical effect on the padding of the element. However, we can use the shorthand approach to simplify our code.

There are some more, even shorter, ways of writing values for `padding`:

When only 3 values are given, the values are interpreted in the following order: top, left & right, bottom.

The following example will make the top padding 10%, the left and right padding 15% and the bottom padding 25%:

```
p { padding:10% 15% 25%; }
```

When 2 values are given, the values are interpreted in the following order: top & bottom, left & right.

The following example will make the top and bottom padding 10%, and the left and right padding 15%:

```
p { padding:10% 15%; }
```

If just 1 value is given, the value is used for all four sides of the box.

The following example will make the padding of each side 15%:

```
p { padding:15%; }
```

Margin

The box's margin works in a similar way to its padding. The following examples are different ways to specify padding for an element, such as the p element:

```
p {
    /* We can specifically give values to any of the
four sides: */
    margin-top:1em;
    margin-right:2em;
    margin-bottom:1.5em;
    margin-left:2.5em;

    /* We can write any of the four types of
shorthand: */
    margin:1em 2em 1.5em 2.5em;
    margin:1em 1.5em 2.5em;
    margin:1em 1.5em;
    margin:1.5em;
}
```

Border

We can specify values for the border's width, style and colour as follows:

```
element { border: width style color; }
```

The following example will give the p element a solid red border with a width of 1 pixel:

```
p { border:1px solid red; }
```

The `border` property will style each of the top, right, bottom and left borders of the element to the given values.

The `border-width` can be specified using any of the standard CSS *measurement units*, such as `px` *(pixels)*, `em` *(em widths)* and `%` *(percentage of parent element's box)*.

The `border-style` affects the way that the border is drawn on the page. It can have any of the following values: `double`, `solid`, `dashed`, `dotted`, `ridge`, `outset`, `groove`, `inset`. Most browsers will display borders with a `solid` border-style by default.

The `border-color` is a 6-digit hexadecimal colour code, as used elsewhere in CSS.

The above code is shorthand. To achieve the same effect, we could have individually specified the 4 sides of the border as follows:

```
p {
    border-top:1px solid red;
    border-right:1px solid red;
    border-bottom:1px solid red;
    border-left:1px solid red;
}
```

For further refinements, we can specify each of the values for each of the sides. For example:

```
p {
    border-top-width:1px;
    border-top-style:solid;
    border-top-color:red;

    border-right-width:1px;
    border-right-style:solid;
    border-right-color:red;

    border-bottom-width:1px;
    border-bottom-style:solid;
    border-bottom-color:red;
```

```
    border-left-width:1px;
    border-left-style:solid;
    border-left-color:red;
}
```

`width` & `height`

We can set the width of an element's content with the `width` property. For example:

```
p { width:75%; }
hr { width:8em; }
img { width:256px; }
```

We may want to specify the widths of our elements so that they fit together well on the page. However, we will not often need to specify a height because the height of a box is usually determined by the quantity and size of text or other content within it.

We can specify height in the same way as width. For example:

```
hr { height:3px; }
img { height:360px; }
```

`width` & `height` Of The Content & Of The Entire Box

Note that `width` and `height` refer to the **content** of the box and not the entire box itself. So, in the following example, the width of the box's content will be 50% and the overall width of the box will be 62%:

```
p {
    width:50%;
    margin:5%;
    padding:5%;
    border:1%;
}
```

The overall width of the box can be calculated as:
content `width` + `padding-left` + `padding-right` + `margin-left` + `margin-right` + `border-left` + `border-right`

For the example above, this calculation becomes:
50% + 5% + 5% + 5% + 5% + 1% + 1% = 62%

Similarly, the overall height of a box can be calculated by considering the `padding`, `margin`, `border` and content `height` within the box.

How `margin`, `padding` & `border` Affect The Width & Height.

We often don't need to specify a `width` or `height` value, since the `margin`, `padding` and `border` properties can be enough on their own.

For example, the following code will style a `p` elements box to have a content width of 50%, even though we have not given a specific value to the `width` property:

```
p {
    margin:25%;
    padding:0;
}
```

In the above example, the `margin-left` will be 25% and the `margin-right` will be 25%. We have specified that there will be zero `padding`. Since the default way for a browser to display a `p` element is without a border, there will also be zero `border`. The entire box will stretch to 100% of its parent element's content width, and so the content width of the `p` element will be 100% – 25% – 25% = 50%.

In this way, we keep things simple by only specifying the smallest amount of properties to achieve the effect we desire. We let the browser work out how to display our elements based on the CSS properties that we specify.

The Box Model Hack

As we have learned, the CSS Box Model uses `width` and `height` to specify the dimensions of the box's **content**, not the entire box.

One problem we face when styling our web pages that a very common browser – Microsoft's Internet Explorer 5 series – mis-interprets the Box Model and uses its own, non-standard box model.

The Microsoft box model interprets the `width` and `height` values as being the width and height of the *entire box*, and not the width of the box's content.

This simple, but fundamental difference makes a real mess of any styles that specify `width` or `height` along with `margin`, `padding` or `border`.

Fortunately, there is a workaround – a code "hack" that forces Internet Explorer to behave properly. For information on this "Box Model Hack", see:
http://www.tantek.com/CSS/Examples/boxmodelhack.html